TRACKING FLIGHT SOFTWARE IN CASSINI MISSION OPERATIONS - USING THE FMT TOOL

Edwin P. Kan and Hal Uffelman

Jet Propulsion Laboratory
California Institute of Technology
4800 Oak Grove Drive
Pasadena, Ca. 91109, USA

(edwin.p.kan@jpl.nasa.gov; hal.uffelman@jpl.nasa.gov)

ABSTRACT

The arduous task of tracking multiple flight software images, across redundant on-board processors / recorders and across time, has been automated and systematized via the use of a unified Flight Software Memory Tracker (FMT) Tool. FMT was developed as part of JPL's Multimission Spacecraft Analysis System, and was customized for the Cassini spacecraft mission. Cassini is designed for eleven years of space flight, over which flight software is expected to be have multiple updates, including several major version revisions. The overall design and operational procedures of FMT are described in this paper.

Keywords: Flight software memory tracking; Cassini spacecraft; Ground analysis software; Mission operations

1. INTRODUCTION

Tracking spacecraft on-board flight software (FSW) images is a vital activity in ground systems and mission operations. This is particularly true for one-of-a-kind spacecraft built for long duration planetary exploration, during which FSW requires frequent maintenance, patches, parameter changes, and occasional complete new version loads. Complete and accurate knowledge of current and past FSW images is essential.

In the history of spacecraft operations at the Jet Propulsion Laboratory (JPL), ground mission operations analysts utilize various degrees of automation, integration of software tools and manual procedures to track FSW. While dynamic memory addresses can only be tracked by full-up hardware and software simulation, static memory addresses, constants, and certain quasi-static parameter addresses are always tracked. For address spaces of interest, an up-to-date FSW image, a FSW image at a specific time in history, and a trend of certain parameters over time, are often the basis for analysis, diagnosis and prognosis.

To automate and systematize such FSW memory tracking, the Flight Software Memory Tracker (FMT) was developed as a unified tool for multimission operations. FMT was customized for the eleven-year (primary) Cassini mission, launched on October 15, 1997 and scheduled to arrive at Saturn in 2004 for a four-year orbital tour of the planet.

FMT is routinely used by Cassini mission operations to track FSW code and parameter address spaces of interest for the AACS (Attitude and Articulation Control Subsystem) and the CDS (Command and Data System). Multiple ground FMT images are maintained. These images are living images which are updated per activity commanded or sequenced on-board the spacecraft.

FMT was developed as part of the Multimission Spacecraft Analysis System (MSAS), under the auspices of the Jet Propulsion Laboratory Multimission Ground Systems Office (Ref. 1, 2).

2. CASSINI's REQUIREMENTS FOR FMT

AACS and CDS, the two major subsystems on the Cassini spacecraft, both have extensive computing facilities. The FSW of each subsystem must be tracked. Both AACS and CDS are equipped with dual-redundant MIL-STD-1750A computers, each with 512K memory, usually containing FSW of the same version / update as the redundant counterpart, and occasionally containing different FSW versions (during code patches, revisions, and during fault situations). The ten scientific instruments on-board Cassini also have various sizes of embedded FSW.

There are also two redundant solid state recorders (SSR), each with a capacity of 2 gigabits. Multiple (4) copies of AACS and CDS FSW memory load images are resident on each SSR. Spacecraft commands affect SSR memory load images and RAM FSW images differently. Parameter tables on RAM images may not be resident on SSR images; nonetheless, there is a determinate relationship between SSR and RAM images. SSR images may be differ from one another during flight because of radiation induced bit flips, hardware or software faults, or during FSW revisions; the differences must be tracked and reconciled.

Counting all FSW images, AACS tallies 10 copies (there are another two additional copies on extended memory). CDS tallies 10 copies. The SSR copies are formatted essentially like the compiled load images, with additional checksums appended to the data records. The actual RAM images are 512K-word images, basically address-value pairs.

Normal spacecraft operations involve: ground uplinks of commands; on-board execution of real-time commands and stored sequences (of commands); and downlink of telemetry. Through this closed loop interaction, FSW code and parameter tables can be updated via uplink commands and verified via memory readout (MRO) telemetry.

Knowing the contents of a single 512K-word image is fairly straight-forward. However, when updates as frequent as once per week, tracking the same image over time can be an arduous task. Over an 11-year mission, retrieving the time history of parameters and memory contents can become intractable, and requires efficient storage of the multiple 512K images. The challenge of tracking Cassini's multiple FSW copies over redundant computers and SSR's becomes overwhelming without a unified tool such as the FMT.

Functional requirements and software requirements for a FMT for Cassini mission operations are detailed in Ref. 3 and 4. Figure 1 shows the context diagram of FMT. The following summarizes FMT functionality:

- Keep current on-ground replicas of on-board FSW images, in code and parameter tables;
- Track all determinable changes due to execution of uplinked or stored commands;
- Process MRO telemetry data, compare with on-ground copies, and reconcile differences;
- Relate RAM and SSR images;
- Analyze, relate, and reconstruct FSW code and parameters for time profiles or snapshots;
- Facilitate uplink commands to recreate known FSW images via memory patches.

The determination of changes due to execution of uplinked or stored commands is implemented outside FMT. The parsing and interpretation of spacecraft commands are performed within the framework of a JPL program set called SEQGEN (Ref. 5), which has been the subject of many technical papers. The adaptation of SEQGEN to FMT command parsing is via the use of custom interface and model files.

3. OVERVIEW OF FMT DESIGN

Figure 1 is the FMT context diagram. Figure 2 is the FMT (Level 1) Data Flow Diagram.

FMT meets the challenge of handling, updating, archiving and analyzing large amounts of data through a novel, yet intuitive, data and file structure design. FMT is implemented as a program set, consisting of multiple "utility" programs, some of which are programs that also call other FMT

utilities. FMT users are best served by executing scripts, such as UNIX scripts, to achieve the higher-level objectives of FMT.

3.1 Data and File Structure

Figure 2 is the FMT (Level 1) Data Flow Diagram. The variety types of data handled by FMT are evident in this figure. Representation of the different data types and data structures is unified, using a common header, data group, and data record structure. Classes of data groups and data records are created, lending itself to the use of property inheritance features of an object oriented programming language such as Java, in which FMT is implemented.

While FMT's software design in Java is detailed in Ref. 6, the classes of data types (hence their corresponding Java classes) are listed here:

.alv	.ealv	.xalv	(SSR "Assisted_Load_File" data classes)
.fmt	.efmt	.xfmt	(RAM FMT data classes)
.amf	.eamf		(Attribute_Model_File classes)
.adb	.eadb		(Attribute_Data_Base classes)
.msk	.emsk		(Data Mask classes)

The utility of, hence class extension and property inheritance between, these data structures can be easily exemplified by the following:

At the time of a load, the RAM .fmt file contains the data records pertaining to the load, i.e., one data group. Each group has a data group header comprised of time (SCET), image type (RAM | SSR | etc.), and $$EOG (an indicator for "end-of-group"). Each data record is structured to show RAM start_address, data values of up to sixteen RAM addresses, and a new line character. The .fmt file has a file header record and $$EOF (an indicator for "end-of-file").

When an update occurs, the update is incorporated in the .fmt file with a new time-stamped data group appended to the previous data group(s). When data groups are appended, the .fmt file takes the nomenclature of .efmt ("enhanced" fmt) file.

For certain update operations, update files with multiple data groups are generated by FMT utilities. Due to the "spatial" nature of FMT, the update groups need to be correlated with specific "spatial" images, e.g. AACS_A computer vs. AACS_B computer. Hence, the nomenclature of a .xfmt ("extended fmt) file, where the data group header takes an extra descriptor (a Software Image Designator, SID), e.g., SCET; RAM; AACS_A. The following examples show the skeleton of a .xfmt file:

```
DATA_FILE_HEADER
1997-070T00:00:00.000; RAM; AACS_A
000670 e511 e522 740a 8a0f 0000 7b04 8a00 1df5 ... 1df2 85ff 0003 7ff0 7ef0
000680 23e8 7ff0 7ef0 2384 7ff0 7ef0 2309 7ff0 ... 7ef0 24a8 7ff0 7ef0 24af
$$EOG
1997-298T12:00:01.001; RAM; AACS_A, AACS_B
038e80 4189 37f7
039366 7dce 000d 0000
$$EOG
$$$EOF
```

All twelve file types (each with unique extension 'xxx') defined above have identical data header, data group, .xxx and .exxx file structure. Data record structures for different data types are different. A RAM fmt data record has been shown as a 17-element record, consisting of an address followed by sixteen 4-nibble hex values. As another example, an .adb data record takes the following form (in single line):

```
038e80,0,A5.6.7,4189 37f7,1997-098T00:00:00.000,0.001,4/8/1997 0:00:00,
      ACL_PARAMETERS,Attitude_Deadband,6,posDB[0],2,float,7DEADBAND,1,0.001,
      rad,0.0005,0.35,,,,,,,,,,
```

Note that all these files are ASCII text files, which can be imported and exported to popular COTS (commercial off-the-shelf) personal computer editor and application programs, making it relatively painless to perform front-end and tail-end processing. FMT is written in Java, which means that its bytecodes are independent of, and hence executable on multiple end-user computing platforms.

3.2 FMT Program Set

The current FMT design has 23 programs, or "utilities", listed as follows:

fmtbeheadeadb	fmtcmdstemgen	fmtconvert	fmtcreate
fmtcs16	fmtdiff	fmtdv2eu	fmteu2dv
fmtextractdes	fmtfilteralv	fmtmaskalv	fmtmasktdc
fmtmaskxfm	fmtmemupdatecmdgen	fmtmerge	fmtmro
fmtquery	fmtrefresheadb	fmtreinit	fmtsort
fmtuser	fmtupdatebycommand	fmtxemerge	

These "utilities" are normally executed in scripts, i.e., "procedures", in order to achieve an overall high level objective of FMT.

Extracted from the User's Guide (Ref. 7), the following UNIX scripts (among other scripts) are now in operation:

Image Initialization:	configured_directory_build.script	amf_build.script
	adb_build.script	ssr_ram_image_build.script
	basic_ssr_ram_image_build.script	
Image Differencing	ram2ram_diff.script	ssr2ssr_diff.script
	ram2ssr_diff.script	ssr2ram_diff.script
File Version Update	eamf_version_update.script	amf_update.script
Update SSR	update_ssr_from_ram.script	extract_data_recors.script
	ram_alf_diff_gen.script	
CDS Checksum	cds_fletcher_calculation.script	cds_checksum_range_extract.script
	fletcher.script	
Map File Processing	awk_map.script	

4. EXPOSITION OF UTILITIES AND PROCEDURES

This section offers detailed insight into the fmtconvert and fmtdiff utilities and the ram2ssr_diff.script, to illustrate how FMT utilities and procedures work. Such application of utilities and procedures follows the footsteps of Mars Observer and Galileo mission operations (Ref. 8 and 9).

4.1 fmtconvert Utility

This utility embodies the data manipulation of FMT, where certain data types are converted to other data types, and where certain files are manipulated and "evaluated". fmtconvert functions more than mere data format conversion. Its usage is as follows:

fmtconvert <parameters>

For single input file <parameter> options, they are:

[-if1 <input file_1 name]
[-itype1 {adb|eadb|ealv|eamf|efmt|emsk|fmt|ldm|mro|xalv|xfmt|alv|amf}]
similarly for -if2 and -itype2
[-of <output file name>]
[-otype {adb|amf|alv|eadb|ealv|efmt|emsk|fmt|msk|tdc|xalv|dfmt|prg}]
[-scet <scet_time>]
[-sid <software image designators, e.g. AACS_A, CDS_B>]

Example 1: fmtconvert -if SSRA_0_AACS.alv -of AACS_A.fmt

Here, the SSRA_0 AACS memory load is converted (from .alv format) into a AACS_A RAM .fmt image (corresponding to a spacecraft sequence in loading AACS_A FSW from the SSRA_0).

Example 2: fmtconvert -if CDS_A.efmt -scet 1998-123T00:00:00.001 -of 98123.fmt

Here, the current RAM image of CDS_A, an .efmt file with multiple data groups (i.e., load image plus multiple updates) is *evaluated* at scet time of year 1998, day 123, time 00:00:001. The output is a .fmt file with a single data group, representing the CDS_A image at the specified scet time.

4.2 fmtdiff Utility

Analysis of different FMT images often requires comparison at specified times, using masks to compare only specified masked sub-images. The usage of fmtdiff is as follows:

fmtdiff <parameters>
where -if1 <input file name>
[-itype1 {efmt|ealv|tdc|alv}]
[-scet1 <scet_time>]
similarly for -if2, -itype2, and -scet2
[-mask]

Example 3: fmtdiff -if1 CDS_A.efmt -scet1 1998-100T00:00:00.001 -if2 CDS_A.efmt -scet2 1998-165T23:59:59.999 -mask -of 98100_165_MSKdiff.txt

Here, the CDS_A .efmt image is evaluated at DOY 1998_100 and 1998_165, producing two .fmt images, 98100.fmt and 98165.fmt. In turn, these two .fmt are compared, i.e., diff'ed, using the 98100.fmt as the mask. Without the -mask option, the complete images of 98100.fmt and 98165.fmt are compared.

4.3 ram2ssr_diff.script

The usage of scripts is no different than the usage of utilities. Normally, scripts contain calls to utilities as well as scripts. Certain utility also call other utilities during compilation. To wit:

```
ram2ssr_diff.script ram.efmt scet1 ssr.elav scet2 ram_image.mask output_file
                    argv[1]  argv[2] argv[3] argv[4]   argv[5]     argv[6]
```

While the script contains substantial error trapping and user interface interaction, only the skeleton of this script is described as follows:

```
#!/bin/csh -f
fmtconvert -if $argv[3] -of tmp.ram1.efmt
fmtsort -if tmp.ram1.efmt -of tmp.ram2.efmt
ram2ram_diff.script $argv[1] $argv[2] tmp.ram2.efmt $arg[4] $argv[5] $argv[6]
```

where:

```
ram2ram_diff.script ram1.efmt scet1 ram2.emft scet2 ram_image.mask output_file
                    argv[1]  argv[2] argv[3] argv[4]   argv[5]     argv[6]
```

which is:

```
#!/bin/csh -f
fmtconvert -if $argv[1] -scet $argv[2] -of tmp.ram1.efmt
fmtconvert -if tmp.ram1.fmt -of tmp.ram1.tdc
fmtmasktdc -tdc tmp.ram1.tdc -msk $argv[5] -of tmp.maskd_ram1.tdc
fmtdiff -if1 tmp.mskd_ram1.efmt -scet $argv[2] -if2 $arv[3] -scet2 $argv[4] -mask
        -of $argv[6]
```

5. ILLUSTRATION OF END-TO-END FMT ANALYSIS

This section contains a flow of procedures (i.e., scripts) that embody the end-to-end FMT analysis. It is divided into five subsections: (i) Data Setup; (ii) Data Update by Command; (iii) MRO Data Processing; (iv) Command Generation; and (v) Data Analysis.

5.1 FMT Data Setup

This step is basically a one-time process, whereby the proper .fmt and .alv images are set up from the FSW compiler load. Data bases containing parameters of interest to mission ops analysts, are established in .amf and .adb files. A typical script to achieve this process is as follows:

```
#!/bin/sh -v
mkdir ~/SETUP; cd ~/SETUP; mkdir 1stGEN; mkdir 2ndGEN
fmtconvert -if aacs_a0.ealv -of 1stGEN/aacs_a.efmt
fmtcreate -if amf_records.ascii -otype eamf -of 1stGEN/aacs.eamf -ver A.5.6.7 -scet 1997-
    298T12:00:01.001
fmteu2dv -if adb_eu_records.ascii -of 1stGEN/adb_dveu_records.ascii
cd 1stGEN; mkdir 2ndGEN
fmtcreate -if adb_dveu_records.ascii -otype eadb -of 2ndGEN/aacs_a.eadb -ver A.5.6.7 -scet
    1997-298T12:00:01.001
fmtconvert -if 2ndGEN/aacs_a.eadb -of 2ndGEN/parameters.efmt
fmtmerge -if1 aacs_a.efmt -if2 2ndGEN/parameters.efmt -of 2ndGEN/aacs_a.efmt
```

5.2 FMT Data Update By Command

This process updates the base .fmt, .adb and other files by parsing spacecraft commands and interpreting the actions which alter FSW code and parameters. A typcial script is as follows:

```
#!/bin/sh -v
mkdir ~/UpdateBYcmd; cd ~/UpdateBYcmd; mkdir 1stGEN
fmtconvert -if ~/SETUP/1stGEN/aacs.eamf -of 1stGEN/97333.amf -scet 1997-333T00:00:00.001
fmtupdatebycommand -if1 1stGEN/97333.amf -sub AACS -if2 aacsPARMch.sasf -start 1997-
    330T00:00:00.001 -end 1997-340T00:00:00.001 -ver A5.6.7 -of 1stGEN/update.xfmt
cd 1stGEN; mkdir 2ndGEN
fmtxemerge -if ../update_edited.xfmt -eamf ../aacs.eamf -scet 1997-330T00:00:00.001 -id .. -od .
```

5.3 FMT Data Processing of MRO Telemetry Downlink

This process updates the current .efmt, .eadb and other files by processing MRO data. A typical script is as follows:

```
#!/bin/sh -v
mkdir ~/MROupdate; cd ~/MROupdate; mkdir 1stGEN
# fmtmro -if aacs.mro -x
fmtmro -if aacs.mro -p -sub=AACS -des=PR -of 1stGEN/aacsPR.mro
cd 1stGEN; mkdir 2ndGEN
fmtconvert -if aacsPR.mro -of mro.xfmt -sid AACS_A
cd 2ndGEN
fmtxemerge -if ../mro.xfmt -eamf ../aacs.eamf -scet 1997-304T13:59:59.999 -id .. -od .
```

5.4 FMT Command Generation

This process involves analyses of .efmt and other images, to determine the differences between actual and desired on-board images, hence generating memory_write commands to implement the updates:

```
#!/bin/sh -v
mkdir ~/CMDgen; cd ~/CMDgen; cd 1stGEN
fmtsort -if aacs_a.efmt -of 1stGEN/sorted.efmt
fmtsort -if 1stGEN/sorted.efmt -of 1stGEN/sortTWICE.efmt
cd 1stGEN; mkdir 2ndGEN
fmtmemupdatecmdgen -if1 sortTWICE.efmt -scet1 1997-365T23:59:59.999 -if2
    sortTWICE.efmt -scet2 1997-300T23:59:59.999 -of 2ndGEN/cmd_97300_365.ascii -target
    AACS -survivor 2
```

```
fmtmemupdatecmdgen -if1 sortTWICE.efmt -scet1 1997-365T23:59:59.999 -if2
    sortTWICE.efmt -scet2 1997-300T23:59:59.999 -of 2ndGEN/cmd_97365_300.ascii -target
    AACS -survivor 1
```

5.5 FMT Data Analysis

A variety of analysis can be performed, including sorting, evaluation, query and diff'ing images. These procedures call utilities such as fmtsort, fmtconvert, fmtmask, fmtmerge, fmtquery and fmtdiff, and other scripts. Their formulation is obvious from the exposition in Section 4 and 5.

6. SUMMARY

The Flight Software Memory Tracker (FMT) is being embraced as a new and powerful ground analysis tool for the Cassini spacecraft mission. The arduous task of tracking multiple copies of FSW images, over physical media and temporal span, has been made tractable via a unified and systematized tool, the FMT.

This paper contains a synopsis of how FMT is designed, what it contains (in terms of "utilities" and "procedures", i.e., "scripts"), and how it achieves an end-to-end processing of commands and telemetry readout updates.

FMT is a multi-mission tool. FMT's software engineering is targeted for generic FSW tracking, and FMT's implementation is in Java. FMT can be run over multiple platforms and is readily extendible to missions other than Cassini.

ACKNOWLEDGEMENT

This work was carried out at the Jet Propulsion Laboratory (JPL), California Institute of Technology, under contract to the National Aeronautics and Space Administration. H. Uffelman was with Arcata Associates, Inc., under contract to JPL. The functional requirements of this work were developed by R. Morillo; engineering design / software specification by E. Kan and H. Uffelman; software design by A. Wax; and implementation by A. Wax, J. Tusynzski and T. White.

REFERENCES

1. Wilson, Robert. K., and M. Hill, M., "Spacecraft Analysis, MSAS (Multimission Spacecraft Analysis System) - A Multi-Mission Solution," Paper #5f001, Proc. SPACEOPS 1998, 5th Int. Symp. on Space Mission Operations and Ground Data Systems, Tokyo, Japan, Jun. 1-5, 1998; also Document #JPL D-9173, Rev. D, (Functional Requirements Document), Jet Propulsion Laboratory, July 10, 1996.
2. Murphy, S.C., et.al., "Customizing the JPL Multimission Ground Data System," Proc. SPACEOPS 1994, 3rd Int. Symp. on Space Mission Operations and Ground Data Systems, held at GSFC, Greenbelt, Md., USA, Nov. 14-18, 1994.
3. Tapia, E. (custodian), "Cassini Functional Requirements 3-291, Uplink Formats & Command Tables," Jet Propulsion Laboratory Document #CAS-3-291, Rev. E, Jan. 24, 1997.
4. Kan, E. P., and H. Uffelman, "CDS (Command and Data Handling Subsystem) Package Requirements Document - Flight Software Memory Tracker," Jet Propulsion Laboratory Document #JPL D-9173 (Section 3.2), July 10, 1997.
5. Salcedo, J., "Advanced Multimission Operations System - Sequence Subsystem (SEQ) Version_19 SEQ_GEN User Guide," Jet Propulsion Laboratory Document #JPL D-111261, also JPL MOSO Document #W4_MSEQ0716-01-00-09, Dec. 1, 1993.
6. Kan, E. P., et.al., "FMT (Flight Software Memory Tracker) for Cassini Spacecraft - Software Engineering Using JAVA," Proc. IASTED '97 Int. Conf. on Software Engineering, San Francisco, CA., USA, Nov. 2-4, 1997.
7. Kan, E. P. and H. Uffelman, "Flight Software Memory Tracker (FMT) User's Guide," Jet Propulsion Laboratory Document #JPL D-9173 (Section 3.2), Sept. 17, 1997.
8. Kan, E. P., "Mission Operations Data Analysis Tools for Mars Observer Guidance and Control," Proc. SPACEOPS 1994, 3rd Int. Symp. on Space Mission Operations and Ground Data Systems, held at GSFC, Greenbelt, Md., USA, Nov .14-18, 1994.
9. Kan., E.P., "Low Bit Rate Autonomous Spacecraft - End-to-End G&C System Design," Proc. AIAA GNC Conf., (American Institute of Aeronautics and Astronautics, Guidance and Control Conference) Paper #96-3925, San Diego, CA, USA, 7/23 -31/96.

Figure 1. Flight_Software_Memory_Tracker (FMT) Context Diagram (Level 0)

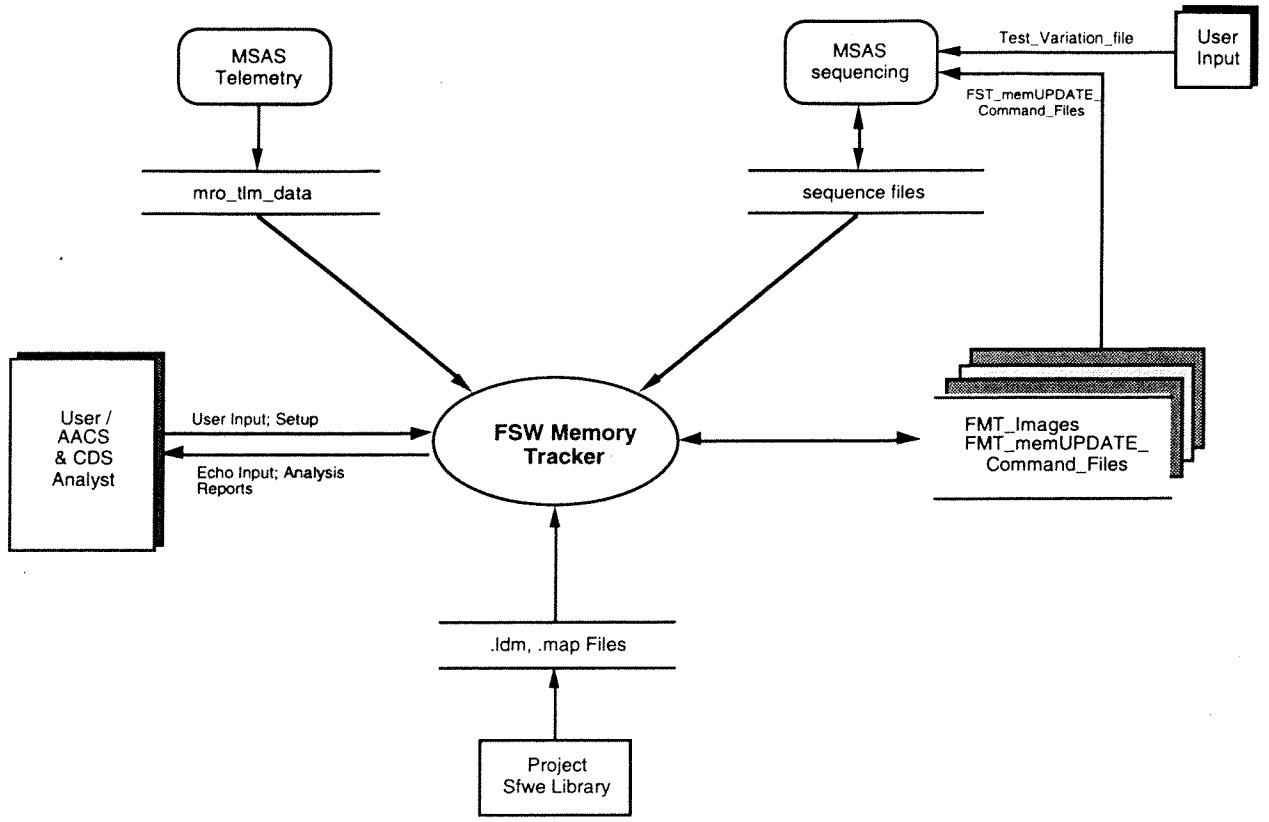

Figure 2. FMT Level 1 Data Flow Diagram